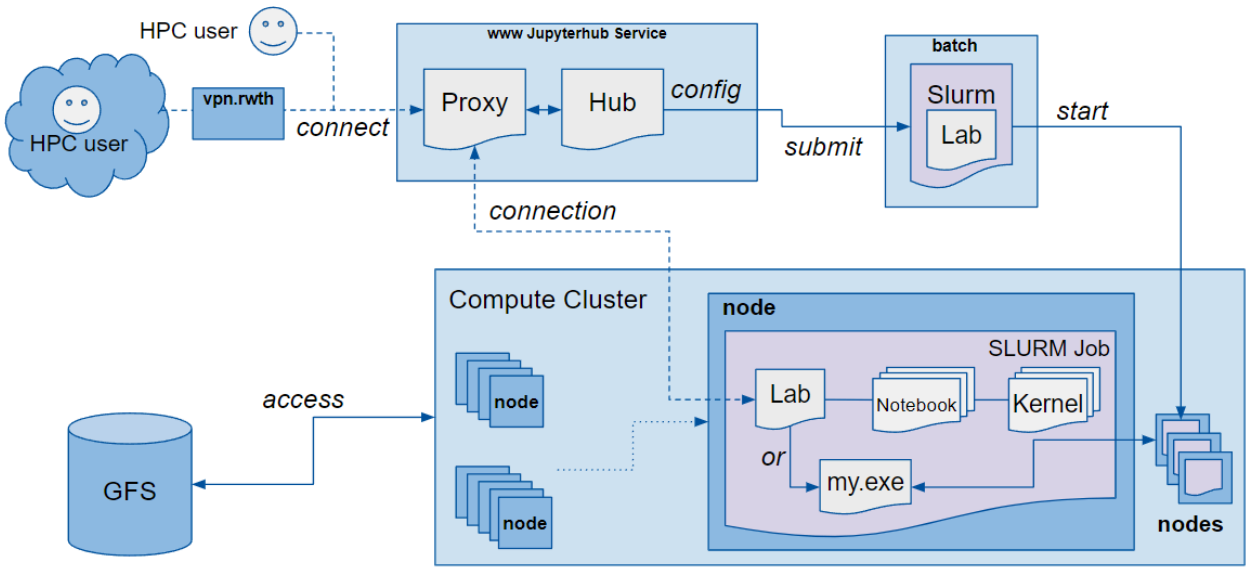
Guidelines and advice for deploying a JupyterHub on High Performance Computing infrastructure

Overview

This document tries to convey context and guidelines for the deployment of JupyterHub instances within HPC environments. These guidelines are only mere recommendations based on experiences managing an HPC JupyterHub combined with Slurm at the RWTH Aachen during a time period of 2 years.
This document assumes therefore that a JupyterHub installation communicates with a SLURM scheduler for submitting classical batch jobs in an HPC environment. HPC sites should adapt their JupyterHub deployments to best suit their specific needs and limitations regardless of the information provided here.
To achieve this, the document provides a small context overview of each relevant concept for a functioning HPC JupyterHub. After each context section, the recommended guidelines are presented. The following diagram gives a reference of a JupyterHub service running within an HPC system. It can be used to supplement the points discussed further into the document.

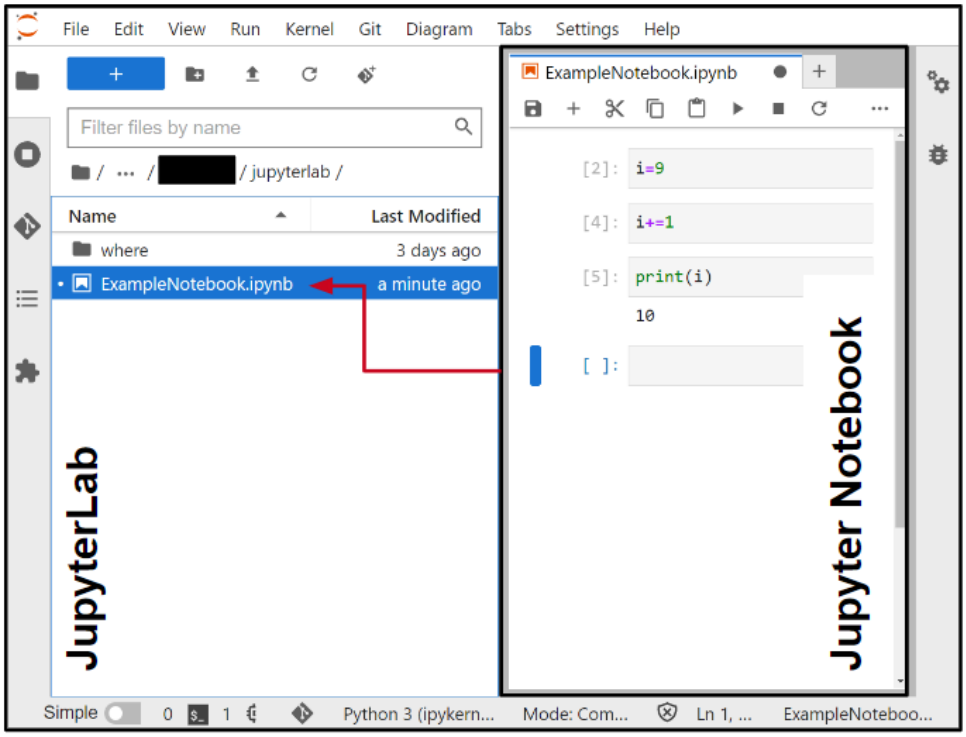


A basic overview is also given in the official JupyterHUb documentation under:
https://jupyterhub.readthedocs.io/en/stable/

Jupyter Infrastructure

Conext

An interactive Jupyter infrastructure for research purposes in HPC requires the three main Jupyter software components: JupyterNotebooks, JupyterLab, and Jupyterhub. Jupyter Notebooks and JupyterLab are part of the same Jupyter ecosystem and are designed to work together for interactive computing by providing user interfaces. Jupyter Notebooks are the original interface that allows users to create and share code, equations, visualizations, and narrative text in the form of notebook files.
They're highly interactive for the single user, letting them write and execute code in various programming languages (most commonly Python). These Jupyter Notebooks are traditionally run in local installations of a Jupyter Notebook instance. JupyterLab is the newer web-based interface for working with Jupyter Notebooks and other file types. It extends the capabilities of the traditional Jupyter notebook interface allowing users to open multiple notebooks, files, and terminals side by side in tabbed views.



The JupyterHub is a server side software that is installed and managed by system administrators that connects to JupyterLabs used by users. JupyterHub handles the authentication, user management, and resource allocation, making it a scalable JupyterLab manager for many users.
JupyterHub is commonly used to accommodate multiple users that need access to distributed computational resources and their Jupyter notebooks. The JupyterHub can therefore be configured to launch a JupyterLab as the user interface. When users log into a JupyterHub,

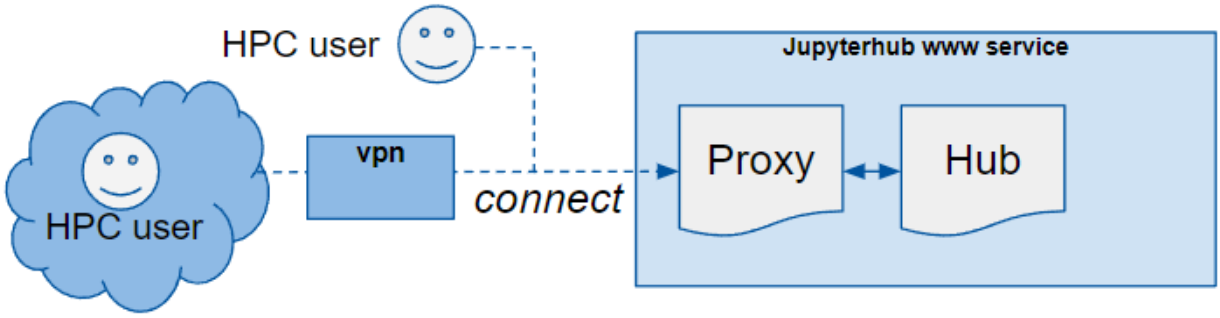
they can launch JupyterLabs to work on their notebooks, scripts, data files, and more. JupyterHub manages the multi-user aspect and server-side operations, while JupyterLab runs as a client interface at the compute nodes.

The Jupyterlab interface can be expanded with plugins that extend its features, integrating with external projects like git to offer version control. User code within a JupyterLab is still stored and edited using Jupyter Notebook files. The code itself runs using kernels and the kernels are installed as separate components and are available within the JupyterLab, but separate from the JupyterHub. The kernels correspond to programming languages and their respective running environments. A kernel runs the moment the user executes a line of code within the Notebook and at the compute node location.

Guidelines

To run JupyterLabs as Slurm Jobs, they first need to be submitted to the Slurm scheduler by the JupyterHub. The Jupyter submission software component is called a **spawner** and runs within the JupyterHub before the user can run any code.
Users need to configure their desired hardware and software environment within the spawner that is also within the JupyterHub. Only then can an appropriate JupyterLab Job be submitted and started by Slurm on a compute node. This is the most problematic and unique part of the entire process. The spawner should be tailored to the hardware partitions and traditional submission rules of the HPC site.
The `https://github.com/jupyterhub/batchspawner` offers the basis for spawners and can be adapted to a Slurm spawner. The unofficial `https://github.com/silx-kit/jupyterhub_moss` also offers an option to submit jobs to existing Slurm installations.
The spawner can be made to work as a filter of allowed Slurm options. In our experience, limiting the visible partitions, quantity of compute resources and runtime limits can be done at the Spawner level with a great deal of extra work. Alternatively, the spawner can be left simple and resource limitations can be left to the Slurm scheduler.
In this second case, if a Spawner JupyterLab job is submitted, but violates limits, the Slurm controller would deny the request and the user would be presented with a submission error. This works very analogously to normal batch job submission, but leaves the error interpretation to the user. A filter of valid options at the Spawner level produces less user support requests. For more on spawner please see:
https://jupyterhub.readthedocs.io/en/latest/reference/spawners.html

Authentication of users happens through the JupyterHub. It is recommended that users first authenticate to a VPN and that the JupyterHubcan only be reachable from within the secure network.



This adds a layer of security to the web service that might be considered a weak security access point to the HPC system. Integration with two-factor authentication systems is also recommended but might be not trivial to implement for an HPC site specific authentication method. For more on security please note:
https://jupyterhub.readthedocs.io/en/stable/explanation/websecurity.html

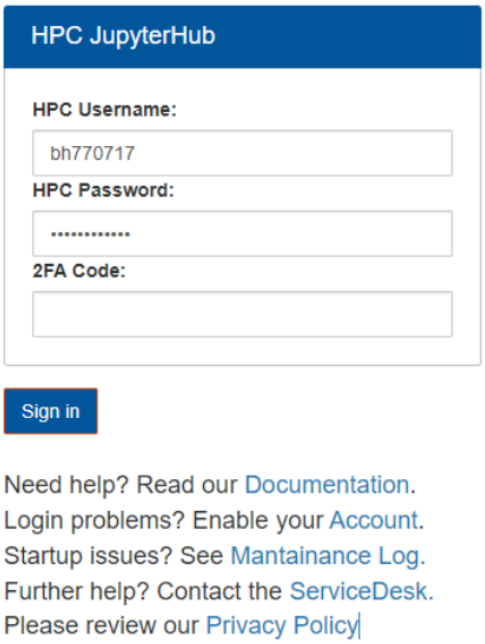
For the main user authentication, local users must exist to let the JupyterHub check credentials. Other options might be available as described in:
https://jupyterhub.readthedocs.io/en/stable/tutorial/getting-started/authenticators-users-basics.html

The authentication might serve all HPC users through the JupyterHub, so enough availability should be guaranteed, on par with any other login nodes. Login systems that offer load-balancing on traditional login nodes might not work on a JupyterHub and therefore see troubles serving the same amount of users through it. For more detail please see:
https://jupyterhub.readthedocs.io/en/latest/reference/authenticators.html

The JupyterHub service should run along a proxy service that handles the connections to the user and allows for the JupyterHub service to be restarted without affecting existing JupyterLab sessions. The Proxy needs the required SSL certificates to allow for a good user experience and be trusted by browsers. A failing Proxy would in the worst case, break all connections to the existing JupyterLab sessions running as Slurm jobs, but would allow for reconnection after restart. A JupyterLab session would therefore only end when the JupyterLab instance itself is terminated by the user or by Slurm. The goal of the proxy is to make running JupyterLabs independent of the JupyterHub. We have seen great success with this method and would only suggest providing distinct hardware for both. For more on networking please see:
https://jupyterhub.readthedocs.io/en/stable/tutorial/getting-started/networking-basics.html

Since the JupyterHub and JupyterLab are two different software installations, it is advisable that both versions are compatible with each other at all times. Forcing the JupyterHub version is trivial, as it is controllable and installed by the admin itself. The JupyterLab runs as a Slurm job, and therefore in user space as a user environment. If liberty is given to users to deploy their own JupyterLab environments, these must be version compatible with the JupyterHub itself.

Startup and communication between them will fail otherwise. Providing read only JupyterLab environment installations forces users to use compatible versions, but makes it difficult for them to change the environments.



Ready only JupyterLab environments can be implemented with NFS, CVMFS and other simple unix permissions. To install environments with JupyterLab a conda installation is a good way. Named conda environments help manage multiple different kernels. A monolithic JupyterLab conda environment can serve multiple different kernels, but makes having multiple versions of the same kernel difficult. It is good practice to install and test the kernels on the compute nodes where they will run, and not on the JupyterHub Server.

HPC Users

Conext:

High-Performance Computing (HPC) users perform tasks that require significant computational power and resources. A typical user of an interactive HPC Jupyter is trying to run new or existing Jupyter notebooks on HPC hardware. The interactive access to the HPC hardware is the primary goal of users. Without access to these HPC hardware, users would just run their own code locally on their personal installations of Jupyter notebooks or JupyterLab.

Users see value in the different Jupyter components, mainly in the user friendly interface and ease of sharing files.
Jupyter Notebooks provide an interactive environment that makes it easier for users to test ideas, visualize results, and document their processes all in one place. Jupyter Notebooks allow HPC users to interactively develop and test their code, making it easier to debug and refine algorithms before running them on the full scale of HPC systems. Jupyter Notebooks can be configured to run on HPC clusters and their distributed nodes, allowing HPC users to take advantage of the existing HPC resources while using the familiar notebook interface. Jupyter Notebooks and HPC file systems also support collaborative features, enabling multiple HPC users to work on the same notebook simultaneously. This is used by institutes, courses and research groups.

Guidelines

A JupyterHub that does not offer quick access to run entire notebooks without big delays, will not see users' activity. Users are expected to prepare and submit batch jobs to HPC systems that they are already familiar with.

To interact with the HPC hardware and the Jupyter interface, users should be familiar with the required authentication methods typical to HPC systems. A typical user might not be familiar with other methods of accessing HPC resources. For them the simplicity of drop down menus in the JupyterHub helps limit choice and is familiar.

File upload through scp or other file transfer protocols might be too complicated for users who prefer web interfaces for computing. In this case the availability of global file systems to the user through the browning section of JupyterLabs is a requirement.

Special care needs to be taken for systems that implement auto mounters that unmount unused paths or do not automatically month common paths until they are accessed through the console. In this case periodic and automated triggers for the auto-mounter system on compute nodes might be necessary. It is also recommended to offer users quick access to support tools from within the JupyterLab and JupyterHub interfaces.

Links to maintenance blogs, support emails and ticket platforms are necessary to reduce the amount of friction between issues and solutions with an HPC Jupyter system. Periodic seminars or introductions to the HPC systems and JupyterHub are also recommended. This is good practice in academic institutions where students start studying twice a year and academic projects such as bachelor, master and doctor thesis are initiated throughout the year. For these users, immediate entry points kickstart the learning process will help them finish their projects on time.

Hardware

Context

Typical High-Performance Computing infrastructure consists of clusters, which are groups of interconnected computers working together to handle distributed large-scale computations.

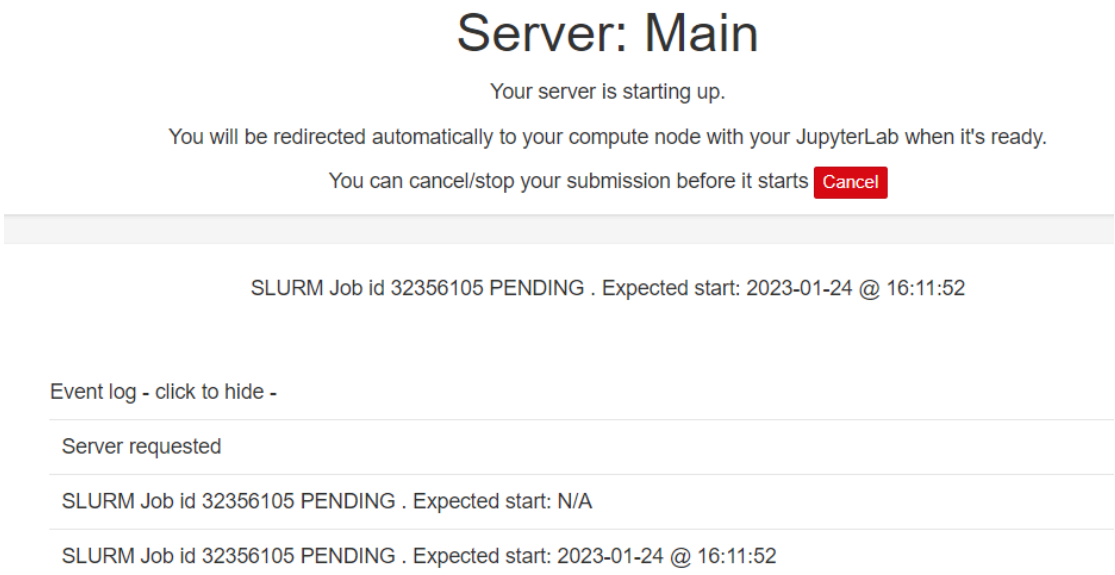
These clusters include various types of nodes, each serving a specific purpose. For instance, login or head nodes act as the entry point for users to log in, submit jobs, and manage their cluster interfaces. Compute nodes perform the actual computational tasks, equipped with enough cpus, memory, and storage to handle intensive processing work. Data transfer nodes specialize in efficient data transfer and storage within the cluster. Fat nodes come with large amounts of memory for handling memory-intensive tasks. Storage nodes are dedicated to storing and retrieving data generated by the compute nodes, often using parallel file systems.

The network in an HPC system connects all these nodes, enabling high-speed data exchange and communication, crucial for the rapid transfer of large volumes of data. VPNs (Virtual Private Networks) provide secure remote access to HPC systems. By using a VPN, the system administrators can ensure that only authenticated and authorized users can access the HPC resources. The VPN enables users to connect to the HPC system from remote locations, ensuring that data transmission is encrypted and protected from unauthorized access. This is particularly important for researchers and professionals who may need to access the system from various parts of the world.

Guidelines

The JupyterHub software will have to run as a service on some sort of physical or virtualized hardware. Virtualized hardware is enough to service the amount of users present on an HPC system through the JuyterHub service. Dedicated physical hardware instead of a virtual JupyterHub machine might also be beneficial to assure easier connectivity to the rest of the cluster. The official guidelines towards load are listed under:
https://jupyterhub.readthedocs.io/en/stable/explanation/capacity-planning.html

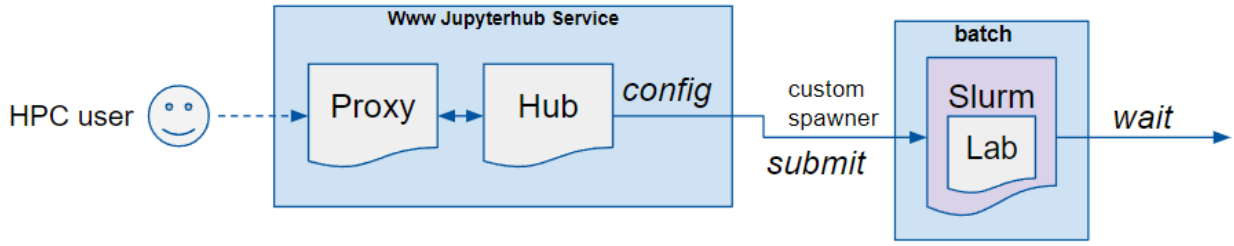
The users need only configure their JupyterLabs and request resources from Slurm when using the JupyterHub. All computations and JupyterLab instances can run on the dedicated compute nodes, where actual load is performed, these are the user requested hardware. The configured hardware should be valid before submission. This can be guaranteed by only providing valid hardware options or by correcting the submission or checking it before handing it to Slurm. Incorrect submissions to non-available or existing hardware will create pending Slurm Jobs that wait indefinitely for hardware.



A proxy service can handle all communication between JupyterLabs and the user. This proxy service can run on the same dedicated physical hardware as the JupyterHub server. A separate machine might be too much for a proxy service, but will help when the main JupyterHub machine needs to go down or restarted. All data manually uploaded to the JupyterLab instance (and therefore the node) runs through the proxy. A good connection through it is therefore desired. In all cases, a good bandwidth connection between the proxy, JupyterHub service and the compute nodes is desired. This facilitates the data transfer from users through the web interface.

The JupyterHub needs only be available on the main web server and can therefore be installed locally. It might need access to global filesystems to read global configurations, such as available JupyterLab installations, user information or data. It should also be able to communicate with Slurm and have a valid Slurm configuration available, since the spawner uses basic Slurm commands like sbatch to generate Slurm jobs.

Traditional login nodes might not be ideal for running the JupyterHub service. Isolating the JupyterHub and having a dedicated JupyterHub login node can help with deployment and changes, without affecting existing login methods. Separating the two also helps to maintain separate configurations to restrict the web service in accordance with the associated risk.



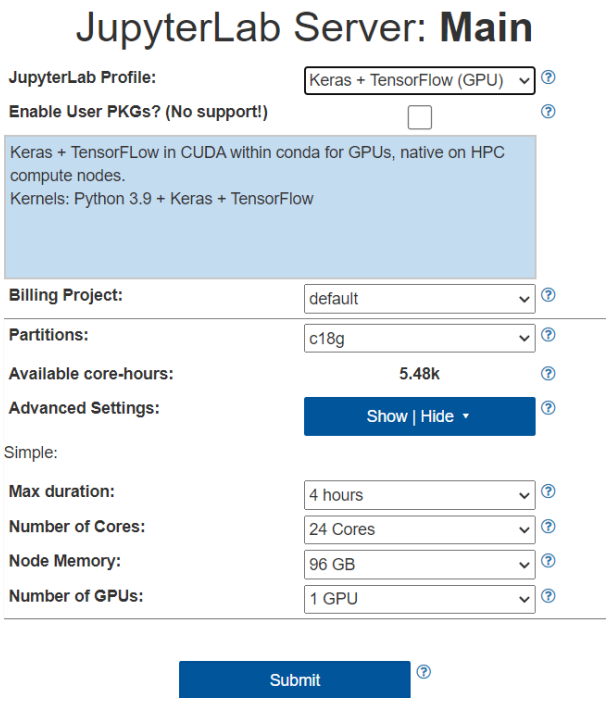
It is important to note that it is preferred to have the user run JupyterLabs as Slurm jobs on compute nodes. The option to run the Jupyter Kernels as Jobs is not desired, due to the inherent lag between executing a command on a Notebook and the eventual submit, queue and running of said command. Once a Jupyterlab is allocated its physical compute resources, a user can use them interactively without delay. This is the desired behavior. For sites that allocate complete nodes for Slurm jobs and do not allow for sharing of resources on a node, a JupyterLab Slurm job might be too inefficient and a waste of resources. For JupyterLab Slurm jobs sharing of nodes is encouraged to avoid allocating hardware that sits idle until users run Kernels. Kernels as Slurm jobs would be more efficient, but not usable in practice by users.

Dedicated Jupyter compute nodes can be set up using a dedicated partition. This partition can be made available to JupyterLab Slurm jobs exclusively by flagging the jobs or by using the submit host as a filter during submission time.

Since users are mostly idle from a computation point of view, hardware can stay at low loads. Further sharing of dedicated hardware can be done by allowing oversubscription of core/gpu's. Slurm traditionally reserves one slot per hardware unit, but this can be overridden in the Jupyter spawner using oversubscription/overcommit to allow for more jobs to use the same hardware unit. Hyperthreading or SMT can be used in this instance to further help increase load.

Because GPUs are expensive and limited in quantity, these should be shared when possible. Nvidia allows for partitioning of modern GPUs using vGPUs or MIG (Multi-Instance GPU) to allow more than one user or job to use the same GPU.
See: https://www.nvidia.com/en-us/technologies/multi-instance-gpu/ for more details. Slurm must be properly configured in these scenarios to allow for all instances to be visible for allocation. Note that not all GPUs support MIG and that MIG splits the available memory on each GPU. This provides less GPU memory to jobs, but services more jobs than an entire GPU would. It is still recommended to have a dedicated partial Jupyter GPU to properly service the interactive needs of users. It is also beneficial to have fully dedicated GPUs available to JupyterLab Slurm jobs. These can serve more memory intensive tasks and are normally also fully utilized by ML libraries like Pytorch, Keras, Tensorflow.

Access to the HPC hardware is typically regulated by each HPC center differently. Access to existing hardware through Jupyter systems can be regulated using the same rules as traditional methods, like Slurm accounts, fair share, backfill and quotas. Users from a Jupyter environment expect fast allocation times, and long queue times discourage the use of the Jupyter interactive environment. A user might be familiar with access to their allowed resources in traditional HPC and expect the same level of access through an HPC Jupyterhub. Hardware only present on specific partitions could be made available to Jupyter users through new exclusive Jupyter partitions. The more hardware the Jupyter user has available, the better can the user workloads be serviced.



Further restrictions can also be established, to safeguard special hardware from inefficient use through JupyterNotebooks. This limit can be enforced by flagging JupyterLab Slurm jobs so that these do not run on protected hardware. This can be done at submit time through Slurm plugins or by limiting the options the user can select in the Jupyter interface.

It is also of note that kernels running on multi-node JupyterLab jobs do not automatically use all available distributed resources. It is up to the user to implement parallel efficient programs that can be called from within their Jupyter Notebooks. The kernels do not get automatically distributed and run on hardware part of the same Slurm job. The parallel job might still block these distributed resources and they could go unused. This should be avoided by forcing only allocations within the same shared memory system or by having users explicitly request distributed Slurm job configurations. In all cases, the resources blocked by users of a running JupyterLab should be “billed” accordingly to discourage idle hardware.

Slurm runtime restrictions can also make yielding hardware resources a problem. This is most relevant when the JupyterLab Slurm job reaches the end of the requested time. In this instance the JupyterLab Slurm job is terminated by the Slurm controller and the user can no longer access their JupyterLab session. Any unsaved progress at this point is lost. It is therefore important to make it clear to users at submission time, that Sessions are terminated and runtimes strictly enforced. This is difficult to change for JupyterLabs only, as changes apply across Slurm for all jobs.